# LING572 Hw4 (kNN)
# Due: 11pm on Feb 6, 2020

The example files are under dropbox/19-20/572/hw4/examples/.

**Q1 (40 points):** Write a script, **build_kNN.sh**, that implements the kNN algorithm. It classifies a test instance x by letting the k nearest neighbors of x vote.

- The learner should treat features as real-valued.

- Use majority vote; that is, each of the k nearest neighbors has one vote.

- The format is: build_kNN.sh training_data test_data k_val similarity_func sys_output > acc_file

- training_data and test_data are the vector files in the text format (cf. **train.vectors.txt**).

- k_val is the value of $k$; i.e., the number of nearest neighbors chosen for classification.

- similarity_func is the id of the similarity function. If the variable is 1, use Euclidean distance. If the value is 2, use Cosine function. **Notice that Euclidean distance is a dissimilarity measure; that is, the longer the distance between two instances is, the more dissimilar (i.e., the less similar) the instances are.**

- sys_output and acc_file have the same format as the one specified in Hw3, and they should include the classification results for both training and test data. When choosing k nearest neighbors for a training instance $x$, one of those neighbors is $x$ itself. Notice that since the other k-1 neighbors could have labels different from that of $x$, the training accuracy could be lower than 100%.

- For each line of sys_output, remember to sort the $(c_i, p_i)$ pairs by the value of $p_i$ in **descending order**.

- Please submit output files correspond to $k = 5, similarity = cosine$.

Run build_kNN.sh with **train.vectors.txt** as the training data and **test.vectors.txt** as the test data. Fill out Table 1 with different values of k and similarity function.

Table 1: Test accuracy using **real-valued** features

| k | Euclidean distance | Cosine function |
|---|---|---|
| 1 | | |
| 5 | | |
| 10 | | |

**Q2 (35 points):** Write a script, rank_feat_by_chi_square.sh, that ranks features by $\chi^2$ scores.

- The format for the command line is: cat input_file | rank_feat_by_chi_square.sh > output_file

- input_file is a feature vector file in the text format (e.g., **train.vectors.txt**).

- The output_file has the format "featName score docFreq". The score is the chi-square score for the feature; docFreq is the number of documents that the feature occurs in. The lines are sorted by $\chi^2$ scores in descending order.

- For $\chi^2$ calculation, treat each feature as binary; that is, suppose the input_file has $a_i$ instances with class label $c_i$. Out of these $a_i$ instances, $b_i$ of them contain the feature $f_k$, then the corresponding contingency table for feature $f_k$ is shown in Table 2.

- Run "cat train.vectors.txt | rank_feat_by_chi_square.sh > feat_list" and submit feat_list.

Table 2: A contingency table for feature $f_k$

|  | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|
| $\bar{f_k}$ | $a_1 - b_1$ | $a_2 - b_2$ | $a_3 - b_3$ |
| $f_k$ | $b_1$ | $b_2$ | $b_3$ |

**Submission:** Submit the following to Canvas:

- Your note file *readme.(txt | pdf)* that includes Table 1 and any notes that you want the TA to read.

- hw.tar.gz that includes all the files specified in dropbox/18-19/572/hw4/submit-file-list, plus any source code (and binary code) used by the shell scripts.

- Make sure that you run **check_hw4.sh** before submitting your hw.tar.gz.